AD-A188 940

NPS52-87-051

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

TERRAIN CLASSIFICATION FROM
DIGITAL ELEVATION DATA USING
SLOPE AND CURVATURE INFORMATION

By

Brenda K. Goodpasture

December 1987

Thesis Advisor: Robert B. McGhee

Approved for public release; distribution is unlimited.

Prepared for:
USACDEC
Ft. Ord, California 93941

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

Kneale T. Marshall
Acting Provost

This thesis is prepared in conjunction with research sponsored in part by contract from the United States Army Combat Developments Experimentation Center (USACDEC) under MIPR ATEC 88-86.

Reproduction of all or part of this report is authorized.

The issuance of this thesis as a technical report is concurred by:

Robert B. McGhee

ROBERT B. MCGHEE
Professor
of Computer Science

Reviewed by:

Vincent Y. Lum

VINCENT Y. LUM
Chairman
Department of Computer Science

Released by:

JAMES M. FREMGEN
Acting Dean of Information and Policy Science

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; Distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): NPS52-87-051	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b. OFFICE SYMBOL (If applicable): Code 52	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: USACDEC; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: ATEC 88-86
8c. ADDRESS (City, State, and ZIP Code): Ft. Ord, California 93941	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
TERRAIN CLASSIFICATION FROM DIGITAL ELEVATION DATA USING SLOPE AND CURVATURE INFORMATION (u)

12. PERSONAL AUTHOR(S)
Goodpasture, Brenda K.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM ____ TO ____	1987 December	57

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Terrain classification; artificial intelligence; Robotics

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Over the past few decades man has concentrated considerable effort in deriving algorithms that can classify terrain in a manner similar to the human visual system. If an implementable algorithm were obtained, man could use this algorithm to add vision to autonomous land vehicles. The applications of autonomous land vehicles are numerous. Movement of large military equipment to previously inaccessible areas and the exploration of unknown areas are examples. The scope of this study is to develop a database from digital elevation data representive of terrain an autonomous land vehicle would traverse and from this database use a two-dimensional algorithm to classify the terrain represented by that data.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Prof. Robert McGhee	22b. TELEPHONE (Include Area Code): (408) 646-2095; 22c. OFFICE SYMBOL: Code 52Mz

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

U.S. Government Printing Office: 1986—606-243

Approved for public release; distribution is unlimited.

Terrain Classification From Digital Elevation Data Using Slope And Curvature Information

by

Brenda K. Goodpasture
Lieutenant, United States Navy
B.S., Eastern Kentucky University, 1982

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

December 1987

Author: Brenda K. Goodpasture
Brenda K. Goodpasture

Approved by: Robert B. McGhee
Robert B. McGhee, Thesis Advisor

Michael J. Zyda
Michael J. Zyda, Second Reader

Vincent Y. Lum
Vincent Y. Lum, Chairman
Department of Computer Science

James M. Fremgen
James M. Fremgen
Acting Dean of Information and Policy Sciences

ABSTRACT

Over the past few decades man has concentrated considerable effort in deriving algorithms that can classify terrain in a manner similar to the human visual system. If an implementable algorithm were obtained, man could use this algorithm to add vision to autonomous land vehicles. The applications of autonomous land vehicles are numerous. Movement of large military equipment to previously inaccessible areas and the exploration of unknown areas are examples. The scope of this study is to develop a database from digital elevation data representive of terrain an autonomous land vehicle would traverse and from this database use a two-dimensional algorithm to classify the terrain represented by that data.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. GENERAL BACKGROUND

A foot soldier standing in a wide valley can survey his environment and choose the best path from his position to a goal. The soldier knows intuitively that in order to reach the peak of a nearby mountain it is to his advantage to traverse a ridge rather than a sheer cliff. Likewise, a tank driver chooses a valley between mountains to reach his goal instead of climbing over one of the mountains to obtain his destination. This study is concerned with the logic humans use in terrain classification and route planning and with the question of whether or not this logic can be converted into computer software so that robots and other machines, such as autonomous land vehicles, can also have this same capability.

The question posed in the above paragraph is a very important one to computer scientists and engineers. With the ability to classify terrain as humans do, autonomous vehicles can be used for various commercial and military purposes. It is with this purpose in mind that millions of dollars have been poured into research involving the use of both digital data and optical data for terrain classification. This study uses digital elevation data to test a new algorithm for terrain classification.

B. ORGANIZATION

A discussion of previous work in terrain classification is presented in Chapter II. This chapter includes different ways of representing data as well as data compression methods. Finally, an overview of various classification schemes of potential value in the

development of the classification algorithm implemented in this study is included. In Chapter III, the specific database used is described. The classification algorithm that is investigated is also developed. In addition, a short description of the computer facilities used is included. Chapter IV describes the structure of the software used. A detailed analysis of the purpose of each major module of code is given. A user's manual for the software is included in this section. Graphics results from the execution of the classification algorithm are presented in Chapter V. Analysis of varying internal thresholds is touched upon as well as performance analysis of the algorithm. Finally, Chapter VI outlines the contributions made by this research and possible extensions into other areas. A list of references and an appendix that contains the code for the algorithm and necessary display graphics are included after Chapter IV.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

Terrain classification has been a subject of interest to scientists for many years. It has applications in the fields of artificial intelligence, hydrology and robotics [1]. Several different models and algorithms have been proposed in these communities [2]. This chapter gives a brief overview of selected ones.

B. REPRESENTATION OF ELEVATION INFORMATION

The most common method for representing surface characteristics is the *contour map*. Paper maps based on this representation have been used for decades and have certainly proven their utility. However, for computational purposes and economy of storage, several alternative digitized methods for representing elevation information have been developed.

The simplest digitized elevation model to manipulate is the *grid digital elevation model [2]*. As its name implies, this model is set up as a grid of elevation points. The elevation of each point in the grid is the only value that needs to be stored since the x,y coordinate of each grid point can be calculated when needed. This model allows any point in the grid to be accessed independently of others and allows links between grid points to be followed if necessary. There are some disadvantages associated with this model. Storing flat areas using this model results in considerable redundancy in the

matrix. If the grid resolution is not small enough, important information can be lost and inaccurate results obtained.

Another digitized model involves the storing of contour lines and is appropriately called the *contour digital elevation model [2]*. This model is more economical in storage and retention of information than the grid digital elevation model. However, as with the paper version of this model, it is difficult to manipulate and access the data.

Models that combine both the advantages of the grid and of the contour digital elevation model have been proposed. Most of these models are based on the *triangular irregular network model [2]*. This model divides a terrain surface into triangles. Flat areas can be divided into large triangles while areas with more relief are divided into smaller ones. These triangular areas are linked using a pointer list. Use of a pointer list involves complex algorithms and increased storage requirements for each data point; therefore, careful selection of data points is necessary to optimize this model. A major disadvantage of this model occurs when a flat area abruptly changes to an area with considerably more relief. It is difficult to maintain the triangular structure necessary for the pointer list in this case. Triangles to join the two areas become long and narrow and these triangles tend to diverge from the actual surface. This disadvantage is highly undesirable in graphics displays of this model. [2]

C. DATA COMPRESSION METHODS

The above section described various models of representing elevation information. These models work well for simple applications, but many applications need a higher

level of abstraction. This section deals with some methods of representing the digitized databases in a more compressed manner.

1. Surface Patches

A common method of data compression used in graphics is *surface patches [3]*. A filled polygon is drawn using selected points in the database. Many polygons joined together form contour features such as mountains. These polygons or patches resemble a "smoothed" version of the terrain represented by the database [4]. This method is used to give a visual display. A major disadvantage with graphics methods is storage of the objects to be displayed. To be able to reproduce the polygons when needed, tables containing the vertices, edge segments and polygon segments must be maintained in memory. The algorithm utilized in this study classifies each data point rather than a collection of data points like those represented by a terrain surface patch. Thus any graphics representation of the terrain classification algorithm results are based on individual points instead of a surface made up of several points. Therefore graphics methods based on surface patches are not used.

2. Octrees

An *octree* is a hierarchical tree structure used to store three-dimensional objects. To generate a node of the octree, a region of three-dimensional space is divided into eight patches or octants. A data element for each of the octants is stored in the node. If the octant is empty, then the individual element of the three-dimensional space or the voxel type is "void." If all voxels of an octant are homogeneous, then the data elements of the node are all the same. A heterogeneous octant is subdivided into octants and

pointers to the next node are stored in the appropriate data elements. This division process continues until all voxels in the octant are homogeneous [3].

Octrees, as with all structures which make use of pointers, are complicated to manipulate and require considerable storage area. Even a small mountain stored in an octree structure can occupy two or three megabytes of memory. Creating objects dynamically from an octree can take hours depending on the graphics method employed. These disadvantages render use of octrees undesirable for most purposes.

3. Skeletal Lines

Skeletal lines are lines of points that show characteristics of the terrain they represent [2]. For example, one way to classify the points in a digital terrain database is to first look for the extremum points. Extremum points are pits or peaks. These are points for which all surrounding neighbors are either higher or lower than the given point. From these points, it is possible to aggregate surrounding points to identify hills and dales. Extremum lines, lines that separate dales (ridge lines) or lines that separate hills (channel lines), can then be identified by numerous algorithms. These extremum lines and extremum points make up the skeletal lines and only the coordinates of the points on these lines need to be recorded in a database of hills and dales. Skeletal lines take up less storage space and are richer in information than a grid digital elevation model.

4. Gradient Lines

Gradient lines, where they exist, are always orthogonal to contour lines. From a database consisting of just gradient lines, a contour map of the terrain represented by the database can be constructed by drawing a series of lines orthogonal to the gradient lines.

However, the reconstructed contour map does not contain the elevation of the contours. This disadvantage of gradient line storage renders it of little value to humans who depend on elevation values for planning purposes.

D. DATA RECONSTRUCTION

The data compression methods discussed above alter digitized databases by deleting data. Graphics techniques rely on smoothing algorithms to approximate the original data for displaying. These smoothing algorithms vary with the graphics method employed. Some algorithms average data points while others selectively pick points depending upon the heuristics defined for the algorithm. By using smoothing algorithms, data points are deleted from the digitized database [3]. The skeletal lines method of compressing data also deletes data from the database, but both methods preserve the general topography of the surface.

E. CLASSIFICATION SCHEMES

Classification of terrain has been an area of intense research [2]. In the following paragraphs, algorithms based on different approaches for solving this problem are presented. Early algorithms were based on various neighborhood comparison schemes. Later algorithms concentrated more on slope and polynomial fits to the terrain surface [2].

1. <u>Neighborhood Comparisons</u>

Algorithms involving general neighborhood comparisons are considered to be weak and unreliable methods for classifying terrain. These techniques "fail because they do not conform to the rules of slope lines" [2]. Simply put, hill climbing via the eight

nearest neighbors is not equivalent to gradient ascent. Most of these algorithms require sorting of the points in the database to find either the lowest or highest ones and from these points, using comparison schemes, peaks or pits are developed depending on the sorting method. Algorithms of this type usually result in erroneously classified points called *artifacts*. Such artifacts render these algorithms undesirable for models that require precision in the classification of terrain.

Neighborhood comparisons that do consider slope in classification of data points have recently been developed. Of particular interest is the *RICHLINE digital elevation model* described by Douglas [2]. This model smooths data before reading it into the program. A neighborhood comparison scheme is then used to locate every data point that is not a ridge. This procedure can also be used to find every data point that is not a channel. From this neighborhood comparison algorithm, a network of thick lines or clouds is obtained. This network is thinned using an algorithm developed by Pavlidis [5]. A skeleton network results from this thinning. The lines of the network are then filtered using Douglas' three-dimensional line reduction algorithm. This model has been proven to be effective in the classification of ridge lines and channel lines. It is more compact in storage than the grid digitized elevation model from which it is derived. Graphical reconstruction of the terrain is possible using spline algorithms. [2]

2. Slope Methods Using Planar Patches

Slope methods using planar patches overcome the problems encountered using the neighborhood comparison methods [2]. Slope is an important consideration in terrain classification. By locating and tracing slope lines from each point upward, it can be determined what peak each point belongs to, thereby defining individual hills. The same

procedure can be done for pits, thereby defining basins. Care must be taken in chosing the slope lines. They must be the lines of greatest inclination through any point.

Algorithms that use real slope lines are proven methods for producing maps showing the hills and basins of the terrain database [2]. However, these algorithms take considerable processing time to locate ridges and channels. Such features are important in terrain classification. Therefore this method is rarely used for this purpose [2].

3. Polynomial Methods

Recent classification algorithms have been based on polynomial fits. Of particular interest to this study is the quadratic approximation of a surface by a least squares fit. Two similar applications of the least squares fit are discussed in this section. The first application is based on a digital picture approach, with the use of gray level analysis to obtain a least squares fit. The second application involves a simulated grid digitized elevation model and a least squares classification algorithm.

Digital pictures obtained by optical scanners or other vision systems contain a wealth of information. By estimating the first- and second-order directional derivatives from the gray level analysis, Haralick is able to classify important features of the surface [6]. Zero-crossings of the first directional derivative categorize peaks, pits, ridges, ravines and saddles. Comparison of the first-derivative with the second-derivative, which determines the curvature of the terrain, finalizes the classification. Haralick also experimented with fitting terrain to a bivariate cubic. While this method shows promise, it needs refinement [6].

The second work of interest is that of Poulos [1]. In this work, a simulated grid digitized elevation model is generated for use in the classification algorithm. A 3 x 3

grid cell that represents the center data point and its eight nearest neighbors is used to classify the center data point by using a least squares fit. Unlike Haralick, Poulos also uses slope to classify terrain. Slope is an important factor in mobility of robots and autonomous land vehicles. Poulos' work is concerned with robots and is the main basis for this study.

Though no known previous work has been performed with B-splines as a classification tool, it appears to be a feasible method. B-splines are a type of bicubic polynomial possessing the special property of first-order and second-order continuity, which is desirable for terrain classification [3]. By approximating a terrain surface to a B-spline, it may be possible to obtain a refined classification of the terrain. This topic is presently in early stages of research at the Naval Postgraduate School.

4. Other Methods

Methods to classify terrain are not limited to the ones discussed in this section. Algorithms are usually designed for a specific purpose. For example, if one were concerned with vegetative cover or soil type in addition to terrain shape, the classification algorithm would be thus modified. Such algorithms are beyond the scope of this study.

F. SUMMARY

Storage and representation of terrain data is an important aspect of terrain classification. In this chapter, most of the widely used methods of storing and representing terrain data have been discussed. As research continues in database storage, new methods will be found to store and maintain terrain data.

Many different algorithms for classifying terrain were presented in this chapter. These ranged from algorithms based on neighborhood comparisons to more advanced ones such as quadratic fits. Research in this area is continually finding more precise algorithms for classification.

The next chapter addresses the problems with terrain classification. A mathematical model for the classification algorithm is presented which includes a simple kinematic vehicle model.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

The main thrust of this study is to build upon the research of Poulos [1] by expanding his algorithm to include additional cases that are presented by Haralick [6] in his digital picture approach, and also to experiment with real rather than synthetic terrain data.

In this chapter, a description of the Fort Hunter-Liggett database, a digitized grid elevation model, is presented. This database is used in experiments involving the proposed algorithm. The mathematical basis for the classification algorithm, as well as the vehicle mobility model and computational resources used in this study, are also included.

B. DESCRIPTION OF DATABASE

The Fort Hunter-Liggett database is stored on the Unix1 VAX system at NPS and is accessible through a program called **make-database-e**. The data in the file is an unformatted sequential file that is arranged as a stream of integers. Each integer of sixteen bits represents both a vegetation code and a bald terrain elevation in feet at a particular point. To make use of the database in this study, the first three bits of the integer are stripped away and the remaining bits are converted to the elevation. Since sampling points are spaced at intervals of twelve and one-half meters, there are 6400 data

points in each kilometer of the database. A more detailed description of the database is given in Smith and Streyle [4].

C. QUADRATIC SURFACE PATCHES

Poulos used a 3 x 3 window with the coordinates of each pixel measured relative to the center pixel. In his approach, a quadratic function of x and y in the form

$$f(x,y) = k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 \tag{3.1}$$

is fitted to the 3 x 3 window. The values of the various coefficients, k_i, appearing in this equation can be obtained solving the equation

$$K = BZ \tag{3.2}$$

where

$$K = (k_1\, k_2\, k_3\, k_4\, k_5\, k_6) \tag{3.3}$$

$$B = \begin{bmatrix} -1/9 & 2/9 & -1/9 & 2/9 & 5/9 & 2/9 & -1/9 & 2/9 & -1/9 \\ -1/6 & 0 & 1/6 & -1/6 & 0 & 1/6 & -1/6 & 0 & 1/6 \\ 1/6 & 1/6 & 1/6 & 0 & 0 & 0 & -1/6 & -1/6 & -1/6 \\ 1/6 & -1/3 & 1/6 & 1/6 & -1/3 & 1/6 & 1/6 & -1/3 & 1/6 \\ -1/4 & 0 & 1/4 & 0 & 0 & 0 & 1/4 & 0 & -1/4 \\ 1/6 & 1/6 & 1/6 & -1/3 & -1/3 & -1/3 & 1/6 & 1/6 & 1/6 \end{bmatrix} \tag{3.4}$$

and

$$Z = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 & z_8 & z_9 \end{bmatrix}. \tag{3.5}$$

The values for Z are the elevations of the data points in the 3 x 3 window. The ordering of these points is important. The first row of the window (starting at the upper-left hand corner) contains z_1, z_2 and z_3. The middle row of the window is comprised of z_4, z_5 (the

center pixel), and z_6. The last row follows in the same pattern. A more detailed explanation can be found in Poulos [1].

Calculation of the slope at the center pixel is easily accomplished since the absolute value of the magnitude of the gradient of the pixel is equivalent to the maximum slope at that point [1]. Thus, since

$$\nabla f = (k_2 \; k_3) \tag{3.6}$$

it follows that,

$$Slope_{max} = (k_2 + k_3)^{1/2}. \tag{3.7}$$

From Eq.(3.1), clearly the Hessian matrix is given by:

$$H = \begin{bmatrix} 2k_4 & k_5 \\ k_5 & 2k_6 \end{bmatrix}. \tag{3.8}$$

The eigenvalues of the Hessian Matrix are required for the classification algorithm described by Poulos. By solving the equation

$$| H - \lambda I | = 0 \tag{3.9}$$

the two eigenvalues, λ_1 and λ_2 are obtained. Classification of the center pixel is then made on the basis of the eigenvalues (the absolute greater of the two is always λ_1) and the slope. The resulting classification is then stored as a property of the center pixel.

Eigenvectors, calculated from the eigenvalues of the Hessian matrix, were not used by Poulos for classification purposes. Haralick, however, uses them in his digital picture approach [6]. This study uses an approach similar to Haralick's to achieve greater precision in classification of ridges and channels in a digital database. Haralick's

technique involves finding the dot product of the principal eigenvector (the eigenvector associated with the largest eigenvalue of the Hessian Matrix) and the gradient at a given pixel [6]. This technique is modified slightly in this study to use the normalized eigenvector and gradient. To calculate the principal eigenvector, v_1, it is necessary to solve the equation

$$\left[H - \lambda_1 I \right] v_1 = B_1 v_1 = 0 \tag{3.10}$$

where

$$B_1 = \begin{bmatrix} 2k_4 - \lambda_1 & k_5 \\ k_5 & 2k_6 - \lambda_1 \end{bmatrix} \tag{3.11}$$

From the first row of the previous equation, a solution for v_1 is clearly

$$v_1 = \left[-k_5 \quad 2k_4 - \lambda_1 \right]. \tag{3.12}$$

If the dot product is below a pre-defined threshold, then the eigenvector is considered to be perpendicular to the gradient. Physically, this implies that the direction of greatest curvature is orthogonal to the direction of greatest slope. This constitutes one way of defining ridges and channels. Mathematically expressed

$$\left| \frac{\nabla f}{|\nabla f|} \cdot \frac{v_1}{|v_1|} \right| \leq \textit{threshold value}. \tag{3.13}$$

The mathematical basis of this study has been established by Poulos and Haralick. The algorithm presented here first classifies pixels by curvature using the eigenvalues. Next, it classifies pixels by calculating the dot-product of the normalized principal

eigenvector and gradient. The results of both of these classifications are stored as properties of the pixel in the database.

D. VEHICLE MOBILITY MODEL

A specific vehicle type was not used in this study. Instead, three slope ranges specified by two slope values are used. These values can be changed to simulate different types of vehicles.

The first value, which is called **safe_slope** in the program, delineates essentially flat terrain from sloped terrain. The second value, which is called **unsafe_slope** in the program, separates terrain with slopes that have been deemed safe for the vehicle from slopes that would be hazardous to the vehicle.

Even though this vehicle mobility model is very simple, it serves to demonstrate the ability of the terrain classification algorithm to appropriately categorize terrain pixels. By changing the slope values to reflect a particular vehicle, those terrain cells negotiable by the vehicle are revealed.

E. COMPUTATIONAL RESOURCES

Poulos used Franz Lisp operating on an ISI Optimum V workstation. This system was found to be much slower than desired [1]. Several versions of Symbolics Lisp Machines have been added to the facilities of the Computer Science Department in the past year and these machines were used to implement the software. Although the various Symbolics machines are similar, most of the coding and testing was done on a Symbolics 3675, in Common Lisp. A color display system on the Symbolics 3675 was used for the

graphics display portion of this study. An in-depth discussion of the software modules is presented in Chapter IV.

F. SUMMARY

An important aspect of this study is the use of a real grid digitized elevation model. Poulos did not have access to a real database and therefore, was forced to simulate one. Simulated terrain does not reveal the full complexity of real terrain.

The mathematical basis for the classification algorithm is developed in this chapter. A simple vehicle mobility model is also presented. From these models three major categories of classification emerge. One is dependent on the curvature of the terrain and another on the constraints of the vehicle mobility model. The slope determines the primary classification. The three possible primary classifications are: level, safe-slope, and unsafe-slope. The curvature of the terrain is described by the eigenvalues of the Hessian matrix. From these the secondary classification is obtained. The seven possible secondary classifications are: peak, depression, ridge, valley, planar, pass and saddle. In addition, separating lines called ridge lines and channels are found from eigenvector and gradient calculations.

IV. IMPLEMENTATION OF CLASSIFICATION METHOD

A. INTRODUCTION

This chapter describes the structure of the software used in the implementation of the terrain classification algorithm. The software is all coded in Common Lisp and uses the *tv* package for graphics routines. At present, this software only executes on the Symbolics 3675 because of the embedded graphics. A short user's guide is also included in this chapter.

B. STRUCTURE OF THE SOFTWARE

The software for this program is located in four separate files. The **grid-utilities.lisp** file contains most of the functions needed to execute the classification algorithm. The **terrain-constants.lisp** file is comprised of various constants used by functions in **grid-utilities.lisp.** The **conversion-factors.lisp** file includes functions that convert from English to Metric and other miscellaneous functions which are also used by functions in **grid-utilities.lisp.** The last file, **test.lisp,** contains graphics functions which display the classified terrain cell. The functions in this file are called from **grid-utilities.lisp.**

The top-level function **run-program** is located in the file **grid-utilities.lisp.** This function, when called with the correct parameters, executes the classification algorithm. The function **run-program** calls several lower-level functions. Figure 4.1 shows the design of the software. The function **make-tc-map** is a macro that makes an array and names it **hunter**. The function **load-tc-attribute** loads the array with a given attribute.

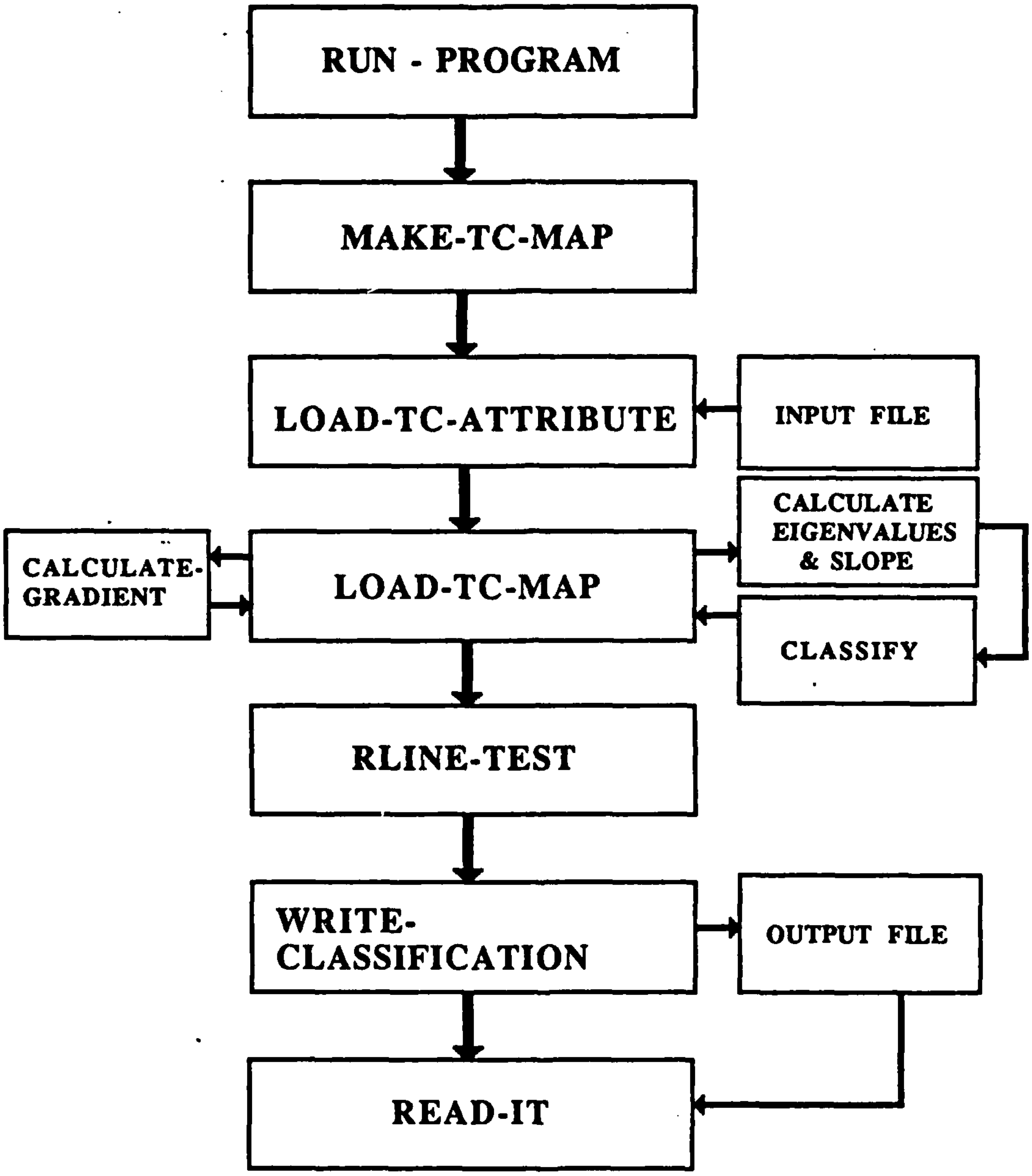

Figure 4.1 Flow Chart of Software

The value associated with the attribute is read in from a designated input file. The function **load-tc-map** is the main lower-level function located in this file. It calls two functions directly and a third one indirectly. These functions are **calculate-eigenvalues-and-slope, calculate-gradient** and **classify.** The first two are self-explanatory by their names. The third function, **classify,** does the terrain classification as described by Poulos1 . The function **load-tc-map** adds several attributes and associated values to the array. The function **rline-test** tests for ridges or channels by taking the dot-product between the normalized principal eigenvector and gradient. This function also adds an attribute-value pair to the array. The function **write-classification** tests for attributes deemed to be important by the user of the software. This function presently considers primary-class, secondary-class and rline as important attributes. An output file is generated by this function. This file contains code for the specific classification of each pixel. The name of this file is a parameter of the function **run-program.** The last function, **read-it,** displays a color picture of the classified terrain cell on the color monitor. It reads codes from the output file for this operation.

The software used in this study is designed in a modular manner. The variable names used are self-explanatory. The four files together occupy less than 30K of memory. The input files for Fort Hunter-Liggett take up 32K and the output files consume approximately 14K. For one grid cell, approximately 75K of storage is needed. A detailed user's guide follows.

C. USER'S GUIDE

To execute the terrain classification algorithm, the *Color* world must be loaded on the Symbolics 3675. As the system is currently set up, this is accomplished by halting the machine and then typing *boot, space, boot* at the FEP command. As the machine cold-boots, it will load the color world. Once this is finished, load the necessary files to execute the algorithm. These files are: **conversion-factors.lisp, terrain-constants.lisp, grid-utilities.lisp** and **test.lisp.** When **test.lisp** is loaded, the machine prompts for the monitor type. Click the mouse on **8-bit high resolution.** Turn the color monitor on if it is not already on. A blue window labeled **Demonstration Window** should appear on the color monitor.

Three parameters are required to execute the program. These are the mapsize, name of the input (data) file and the name of an output file. The mapsize represents the square-root of the number of data points in the input file. This number must be an integer. When using the Fort Hunter-Liggett database, mapsize is 10 for low-resolution and 80 for high-resolution. The input file can be any file from the Fort Hunter-Liggett database or one that is purely simulated. The output file is used for quick displays of the graphics portion of the algorithm. Once an input file has been classified by the algorithm, a code for each terrain feature is written into the output file. By using the function **display** found in **test.lisp,** the classified color map can be displayed on the color terminal with just the parameters mapsize and output file. This saves considerable time when demonstrating the ability of the terrain classification algorithm.

A useful function included in **grid-utilities.lisp** is **get-tc.** This function retrieves the attribute and value list of a particular location in the array. To view the list, simply type

the function name followed by the array name (which is always **hunter** in this program) and the x,y coordinates of the point (each can only be between 0 and mapsize). This function can only be used after the entire program has been executed. The array is erased when the machine is booted.

D. SUMMARY

A discussion of the functions used the terrain classification algorithm has been presented in this chapter. Modularity was a key issue in the design of the software since this study is just a small part of ongoing research in this area at NPS. The user's guide gives the exact details on the operation of the algorithm. Chapter V presents typical results expected from the classification algorithm and explains the meaning of the different color codings.

V. RESULTS

A. INTRODUCTION

The classification algorithm developed in Chapter III describes a surface by a quadratic least squares fit to a 3 x 3 window about each pixel. The algorithm also tests for ridge lines and channels by taking the dot product of the normalized principal eigenvector and gradient. If the dot product is below a pre-determined threshold, then the pixel is classified as either a ridge or a channel. The mathematical basis for this algorithm has previously been established by Poulos [1] and Haralick [6].

Poulos tested his algorithm on several terrain scenes. While these scenes were machine-generated, they proved that the algorithm consistently classified major terrain features correctly. In this chapter, two separate real-life terrain scenes are classified by the proposed algorithm. The results are displayed on the Symbolics Color Monitor.

B. TYPICAL MAPS

The use of the Symbolics Color Monitor to display classified terrain cells offers a great improvement over the bitmaps used by Poulos. By color coding the various terrain categories, a more readable map is obtained.

Seven basic colors are used for the different terrain categories. Since a pixel could possibly be classified in more than one category, the order of testing for the attribute values decides which category is displayed. This order can be altered. All scenes in this

chapter have the same testing order. The table below states the correspondence between the pixel color and its terrain category.

TABLE 5.1

CLASSIFICATION OF TERRAIN PIXELS BY COLOR CODE

COLOR	CATEGORY
White	Edge
Red	Unsafe-slope
Blue	Channel
Black	Ridge
Yellow	Planar
Green	Flat
Orange	Safe-slope

Figure 5.1 contains two separate examples of classified terrain cells as displayed on the Symbolics Color Monitor. In conformity with the above table, the ordering for testing of attribute values in these cells is: edge, unsafe-slope, flat, ridge, channel, planar, and safe-slope. As can be seen, Figure 5.1(a) consists mainly of unsafe-slope pixels. A ridge line and several peaks (flat areas surrounded by unsafe-slope) are also visible in the center of the picture. A channel beginning in the upper right hand corner of the picture and moving diagonally across it is also visible. Figure 5.1(b) also reveals several peaks and slight traces of ridges. Both pictures have areas where many terrain features are side by side without a specific pattern. These areas do not contain major terrain features. Slight abnormalities in the terrain surface and inaccurate elevation readings are possible reasons for the classification of these pixels. By "smoothing" the data before analyzing it, many of these artifacts can be made to disappear. Increasing the curvature threshold causes more pixels to be classified as flat. However, the value for the threshold used in this study represents a 10 foot change in elevation over a distance of thirty feet. This

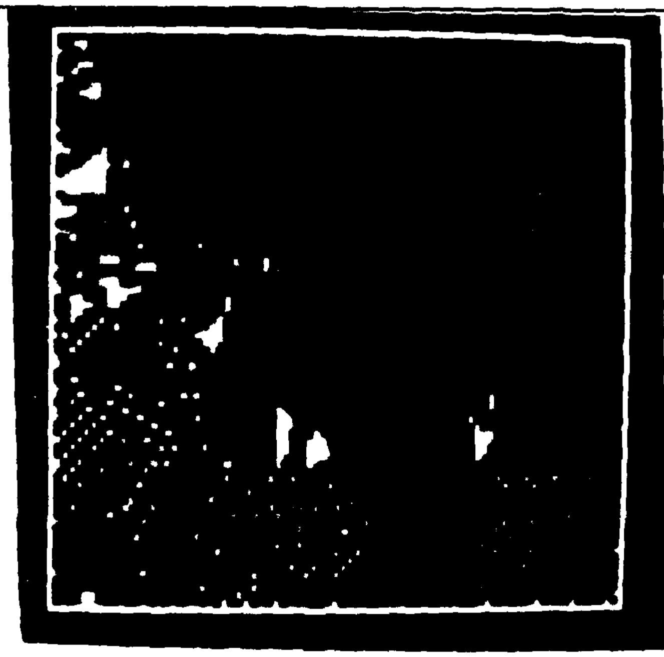

Figure 5.1a Terrain Cell Map

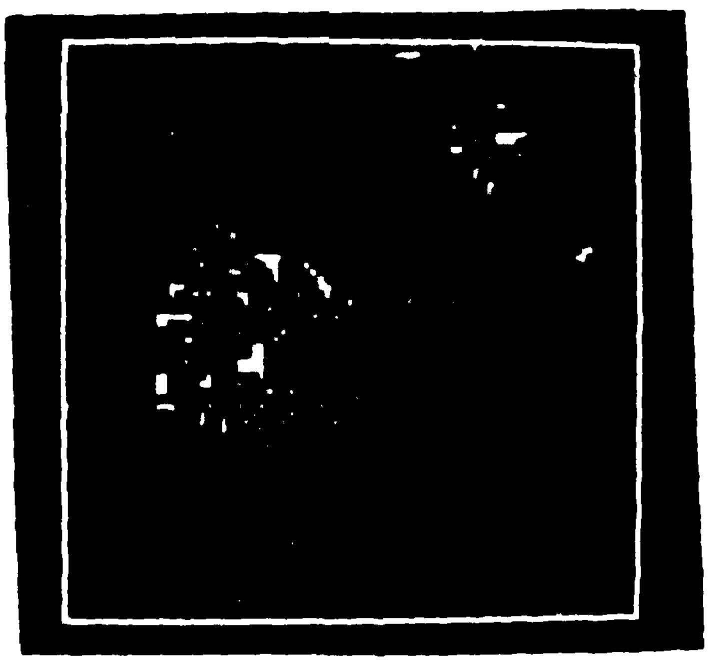

Figure 5.1b Terrain Cell Map

Figure 5.1 Classified Terrain Cell Maps

represents substantial curvature. Further work in this area is now in progress by other students at NPS.

C. PERFORMANCE EVALUATION

Program modules are designed to do very specific tasks and can be used in other programs. Because of the extent of the modularity, the program takes nearly six minutes to execute. By compiling the program and re-arranging the code, the execution time can be reduced by half. Poulos' algorithm, executing on the ISI in Franz Lisp, took 20 minutes to classify a simulated terrain cell of 900 data points [1]. By comparison, this algorithm takes a maximum of six minutes and classifies 6400 data points. Evidently, use of a Lisp Machine has resulted in a significant time savings which is important in time critical uses of this algorithm.

D. SUMMARY

This chapter presented typical examples of classified terrain cells. The use of color to represent the classified terrain cells has significantly increased their readability. This was a problem in previous work. In addition to improvements in readability, an improvement in performance has also been achieved in this study. Specifically, in terms of pixels processed per second, a speed up factor of roughly 25 has been achieved by using a Lisp Machine as a host rather than a general purpose workstation. The next chapter summarizes the major results of this study and also explores future extensions of this work.

VI. SUMMARY AND CONCLUSIONS

A. RESEARCH CONTRIBUTIONS

This study has made several advances over previous work. The use of a Lisp Machine to execute the classification algorithm is an important one. A Lisp Machine is specially designed to execute Lisp code. For this reason, the execution time of the classification algorithm has been significantly reduced.

A different method for finding ridges and channels that had not previously been applied to a grid digital elevation model was introduced in this study. By taking the dot-product between the normalized principle eigenvector and gradient, pixels not previously classified as a ridge or channel could be identified as such if they were below the curve threshold. This results in a more precise classification.

As a result of this study, software on terrain classification written in Common Lisp and residing on the Symbolics 3675 now exists. It can be used as a basis for further research or incorporated into library functions.

B. RESEARCH EXTENSIONS

Research in terrain classification is important to many fields of study. Artificial intelligence programs involved in route planning can make use of the terrain classification algorithm in computing costs for an A^* search [7]. Watersheds can be identified by classifying the terrain to reveal ridges and channels. This is an important topic in hydrology [2]. The applications are numerous.

Future research at NPS should involve the use of B-splines as a method of smoothing the data before analyzing it. Other representations of the terrain surface such as reconstructed color contour maps, possibly in three-dimensions, should also being explored. The possibility of using a digitized picture in conjunction with the grid digital elevation model to refine terrain analysis as well as identify cultural features buildings is another area which ought to receive consideration. It is hoped the the work of this study will encourage such investigations.

APPENDIX - LISP CODE FOR THE TERRAIN CLASSIFICATION ALGORITHM

```
;*********************************************************************************************
;* LIST OF FUNCTIONS IN THIS FILE
;*
;* 1. run-program - top function to run the digital terrain classification - inputs are
;*    mapsize which is the size of the array from the datafile, i.e. for high resolution
;*    the array would be 80 x 80, so mapsize would be 80. The array must be a square to
;*    work in this program. The datafile is the input file and it must be in double
;*    quotes. The classfile is the output file and it also must be in double quotes.
;* 2. make-tc-map - makes an array with the name hunter and of the size given by the
;*    user.
;* 3. load-tc-attribute - loads the array from the datafile with the slot given. In the
;*    case of this program, it loads elevation, but other slots could be loaded.
;* 4. load-tc-map - loads the array with all necessary mathematical data and also
;*    primary and secondary classifications according to the eigenvalues of the Hessian
;*    matrix. In this example, the array is called hunter but it could be changed.
;* 5. rline-test - uses data from the array by accessing it's property list. From this
;*    it determines if the pixels is a ridge or ravine line. This information is then
;*    placed in the property list.
;* 6. write-classification - takes the array and outputs the desired information to the
;*    classfile which is designated by the user. The output is in bitmap form and may
;*    be changed to represent any desired data that the user wants.
;* 7. get-tc - this function allows the user to see the property list associated with
;*    particular spot in the array. The user must input the terrain-cell-map, in this
;*    case, hunter and the x-coordinate and the y-coordinate of the desired spot.
;* 8. get-tc-group - this function allows the user to see a specific list associated
;*    with a give property of a 3 x 3 square of the array. The terrain-cell-map is
```

```
;*   again named hunter and the x-coordinate/y-coordinate are of the middle cell of the
;*   3 x 3 square.
;* 9. edge-tc-p - tests for edge pixels
;*10. corner-tc-p - tests for corner pixels
;*11. range-tc-p - tests to see if the y-coordinate and x-coordinate are within the size
;*   of the array.
;*12. calculate-eigenvalues-and-slope - this function is used by the load-tc-map
;*   function. It's inputs are five of the constants of the quadratic equation and it
;*   outputs a property list that is appended to each spot in the array.
;*13. classify - this function is called by load-tc-map and it classifies the pixel
;*   according to the eigenvalues and the slope. This information is the appended to
;*   the property list in each spot in the array.
;*14. calculate-gradient - this function is called by load-tc-map and it calculates the
;*   gradient which is appended to the property list in each spot in the array.
;**************************************************************************************

(defun run-program (mapsize datafile classfile)
(make-tc-map hunter mapsize)
(load-tc-attribute hunter datafile 'elevation)
(load-tc-map hunter)
(rline-test hunter)(write-classification hunter classfile)(read-it mapsize classfile))

(defmacro make-tc-map (terrain-cell-map mapsize)
  (list 'setq terrain-cell-map (list 'make-array (list 'list mapsize mapsize))))

(defun load-tc-attribute (terrain-cell-map datafile slot)
  (setq input-stream (open datafile :direction :input))
  (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (do ((xcoord 0 (1+ xcoord)))
```

```
((= xcoord mapsize))
(do ((ycoord (1- mapsize) (1- ycoord)))
    ((< ycoord 0))
 (setf (aref terrain-cell-map ycoord xcoord)
      (cons slot (cons (read input-stream) (aref terrain-cell-map ycoord xcoord))))))
(close input-stream))

(defun load-tc-map (terrain-cell-map)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (do ((xcoord 0 (1+ xcoord)))
    ((= xcoord mapsize))
   (do ((ycoord 0 (1+ ycoord)))
       ((= ycoord mapsize))
    (cond
       ((edge-tc-p mapsize xcoord ycoord)
        (setf (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord)
           (append '(gradient1 egde gradient2 edge
                   primary-class edge eigenvalue1 edge eigenvalue2 edge
                   slope edge eigenvector1 edge eigenvector2 edge)
                 (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord))))
       (t (let* ((z-matrix (math:transpose-matrix
                         (make-array '(1 9) :initial-contents
                               (list (mapcar 'feet-to-meter
                                        (car (get-tc-group terrain-cell-map
                                        xcoord ycoord 'elevation)))))))
              (k-matrix (math:multiply-matrices b-matrix-u z-matrix)))
         (setf (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord)
              (append (classify (calculate-eigenvalues-and-slope
```

```
                    (aref k-matrix 1)(aref k-matrix 2)
                    (aref k-matrix 3)(aref k-matrix 4)
                    (aref k-matrix 5)))
                   (aref terrain-cell-map
                         (- (1- mapsize) ycoord) xcoord)))
         (setf (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord)
               (append (calculate-gradient (aref k-matrix 1)(aref k-matrix 2))
                   (aref terrain-cell-map
                         (- (1- mapsize) ycoord) xcoord))))))))))

(defun get-tc (terrain-cell-map xcoord ycoord)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (cond
   ((range-tc-p mapsize xcoord ycoord)
    (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord)))))

(defun get-tc-group (terrain-cell-map xcoord ycoord slot)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (cond
   ((and (not (edge-tc-p mapsize xcoord ycoord))
         (range-tc-p mapsize xcoord ycoord))
    (list
     (cons (getf (aref terrain-cell-map (- (1- mapsize) (1+ ycoord)) (1- xcoord)) slot)
     (cons (getf (aref terrain-cell-map (- (1- mapsize) (1+ ycoord)) xcoord) slot)
     (cons (getf (aref terrain-cell-map (- (1- mapsize) (1+ ycoord)) (1+ xcoord)) slot)
     (cons (getf (aref terrain-cell-map (- (1- mapsize) ycoord) (1- xcoord)) slot)
     (cons (getf (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord) slot)
     (cons (getf (aref terrain-cell-map (- (1- mapsize) ycoord) (1+ xcoord)) slot)
```

```
        (cons (getf (aref terrain-cell-map (- (1- mapsize) (1- ycoord)) (1- xcoord)) slot)
        (cons (getf (aref terrain-cell-map (- (1- mapsize) (1- ycoord)) xcoord) slot)
        (cons (getf (aref terrain-cell-map (- (1- mapsize) (1- ycoord)) (1+ xcoord)) slot)
              nil)))))))))))))


(defun edge-tc-p (mapsize xcoord ycoord)
  (or (= xcoord 0)
    (= ycoord 0)
    (= xcoord (1- mapsize))
    (= ycoord (1- mapsize))))


(defun corner-tc-p (mapsize xcoord ycoord)

  (or (and (= xcoord 0) (= ycoord 0))
    (and (= xcoord 0) (= ycoord (1- mapsize)))
    (and (= ycoord 0) (= xcoord (1- mapsize)))
    (and (= ycoord (1- mapsize))
         (= xcoord (1- mapsize)))))


(defun range-tc-p (mapsize xcoord ycoord)
  (and (and (>= xcoord 0) (< xcoord mapsize))
     (and (>= ycoord 0) (< ycoord mapsize))))


(defun calculate-eigenvalues-and-slope (k2 k3 k4 k5 k6)
  (let* ((a (+ (* 2 k4)(* 2 k6)))(b (sqrt (- (* (- (* -2.0 k4)(* 2.0 k6))(- (* -2.0 k4)
```

```
                                                        (* 2.0 k6)))
                                   (* 4.0 (- (* (* 2.0 k4)(* 2.0 k6))(* k5 k5))))))
        (eigenvalue1 (/ (+ a b) 2.0))(eigenvalue2 (/ (- a b) 2.0)))
  (setq slope (/ (* 360.0 (atan (sqrt (+ (* k2 k2)(* k3 k3))) 1.0)) 2.0 pi))
  (cond ((< (abs eigenvalue1)(abs eigenvalue2))
         (setq temp 0)(setq temp eigenvalue1)(setq eigenvalue1 eigenvalue2)
         (setq eigenvalue2 temp)))
  (let* ((B11 (- (* 2 k4) eigenvalue1))(number 0)(B12 (-  0 k5)))
   (setq normalized-eigenvector (sqrt (+ (* B12 B12)(* B11 B11))))
   (cond ((= normalized-eigenvector 0)
          (list 'eigenvalue1 (eval eigenvalue1) 'eigenvalue2 (eval eigenvalue2)
                'slope (eval slope) 'eigenvector1 (eval number)
                'eigenvector2 (eval number)))
         (t (setq eigenvector1 (* (/ 1 normalized-eigenvector) B11))
            (setq eigenvector2 (* (/ 1 normalized-eigenvector) B12))
            (list 'eigenvalue1 (eval eigenvalue1) 'eigenvalue2 (eval eigenvalue2) 'slope
              (eval slope) 'eigenvector1 (eval eigenvector1)
              'eigenvector2 (eval eigenvector2)))))))


(defun classify (k)
 (let ((E1 (second k))(E2 (fourth k))(slope (sixth k)))
   (cond ((< slope slope-limit)
          (cond ((and (< E1 (- curv-threshold))
                      (< E2 (- curv-threshold)))
                 (append '(primary-class level seconday-class peak) k))
                ((and (> E1 curv-threshold)
                      (> E2 curv-threshold))
```

```
          (append '(primary-class level secondary-class pit) k))
         ((and (< E1 (- curv-threshold))
               (< (abs E2) curv-threshold))
          (append '(primary-class level secondary-class ridge) k))
         ((and (> E1 curv-threshold)
           (< (abs E2) curv-threshold))
          (append '(primary-class level secondary-class ravine) k))
         ((and (< E1 (- curv-threshold))
               (> E2 curv-threshold))
          (append '(primary-class level secondary-class saddle) k))
         ((and (> E1 curv-threshold)
               (< E2 (- curv-threshold)))
          (append '(primary-class level secondary-class pass) k))
         (t
          (append '(primary-class level secondary-class flat) k))))
 ((> slope unsafe)
  (append '(primary-class unsafe-slope) k))
 (t
  (cond ((and (< E1 (- curv-threshold))
               (< E2 (- curv-threshold)))
          (append '(primary-class safe-slope secondary-class peak) k))
         ((and (> E1 curv-threshold)
               (> E2 curv-threshold))
          (append '(primary-class safe-slope seconday-class pit) k))
         ((and (< E1 (- curv-threshold))
               (< (abs E2) curv-threshold))
          (append '(primary-class safe-slope seconday-class ridge) k))
         ((and (> E1 curv-threshold)
```

```
            (< (abs E2) curv-threshold))
          (append '(primary-class safe-slope secondary-class ravine) k))
         ((and (< E1 (- curv-threshold))
            (> E2 curv-threshold))
          (append '(primary-class safe-slope secondary-class saddle) k))
         ((and (> E1 curv-threshold)
            (< E2 (- curv-threshold)))
          (append '(primary-class safe-slope secondary-class pass) k))
          (t
           (append '(primary-class safe-slope secondary-class planar) k)))))))


(defun write-classification (terrain-cell-map classfile)
 (setq output-stream (open classfile :direction :output))
  (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
   (do ((ycoord 0 (1+ ycoord)))
       ((= ycoord mapsize))
     (do ((xcoord 0 (1+ xcoord)))
        ((= xcoord mapsize))
       (terpri output-stream)
       (let ((first-class (getf (aref terrain-cell-map ycoord xcoord) 'primary-class))
          (slope (getf (aref terrain-cell-map ycoord xcoord) 'slope))
          (second-class (getf (aref terrain-cell-map ycoord xcoord) 'secondary-class))
          (r-line (getf (aref terrain-cell-map ycoord xcoord) 'rline))
          (eigenvector1 (getf (aref terrain-cell-map ycoord xcoord)
                                        'eigenvector1))
          (eigenvector2 (getf (aref terrain-cell-map ycoord xcoord)
                                        'eigenvector2)))
```

```
(cond
 ((equal first-class 'level)
  (cond ((equal second-class 'peak)(prin1 '1 output-stream))
        ((equal second-class 'pit)(prin1 '2 output-stream))
        ((equal second-class 'ridge)(prin1 '1 output-stream))
        ((equal second-class 'ravine)(prin1 '2 output-stream))
        ((equal second-class 'saddle)(prin1 '1 output-stream))
        ((equal second-class 'pass)(prin1 '2 output-stream))
        ((equal second-class 'flat)(prin1 '3 output-stream))
        (t
         (prin1 '8 output-stream))))
 ((equal first-class 'safe-slope)
  (cond ((equal second-class 'peak)(prin1 '1 output-stream))
        ((equal second-class 'pit)(prin1 '2 output-stream))
        ((equal second-class 'ridge)(prin1 '1 output-stream))
        ((equal second-class 'ravine)(prin1 '2 output-stream))
        ((equal second-class 'saddle)(prin1 '1 output-stream))
        ((equal second-class 'pass)(prin1 '2 output-stream))
        ((equal second-class 'planar)(prin1 '4 output-stream))
        ((equal r-line 'ridge)(prin1 '1 output-stream))
        ((equal r-line 'ravine)(prin1 '2 output-stream))
        (t
         (prin1 '7 output-stream))))
 ((equal slope 'edge)(prin1 '5 output-stream))
 ((equal first-class 'unsafe-slope)(prin1 '6 output-stream))))))
 (close output-stream))
```

```
(defun calculate-gradient (k2 k3)
 (let ((slope (/ (* 360 (atan (sqrt (+ (* k2 k2)(* k3 k3))) 1.0)) 2.0 pi))(number 0))
  (cond
   ((> (abs slope) 0)
    (let ((gradient1 (* (/ 1 slope) k2))(gradient2 (* (/ 1 slope) k3)))
        (list 'gradient1 (eval gradient1) 'gradient2
            (eval gradient2))))
   (t
    (list 'gradient1 (eval number) 'gradient2 (eval number)
         )))))

(defun rline-test (terrain-cell-map)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (do ((xcoord 0 (1+ xcoord)))
       ((= xcoord mapsize))
   (do ((ycoord 0 (1+ ycoord)))
        ((= ycoord mapsize))
       (cond
        ((edge-tc-p mapsize xcoord ycoord)
         (setf (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord)
               (append '(rline edge)
                      (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord))))
               (t (let ((gradient1 (getf (aref terrain-cell-map ycoord xcoord)
                                    'gradient1))
                (gradient2 (getf (aref terrain-cell-map ycoord xcoord)
                                    'gradient2))
                (eigenvector1 (getf (aref terrain-cell-map ycoord xcoord)
                                        'eigenvector1))
```

```
(eigenvector2 (getf (aref terrain-cell-map ycoord xcoord)
                    'eigenvector2))
(primary-class (getf (aref terrain-cell-map ycoord xcoord)
                    'primary-class))
(eigenvalue1 (getf (aref terrain-cell-map ycoord xcoord)
                    'eigenvalue1)))

 (cond
((equal primary-class 'safe-slope)
(cond
 ((< eigenvalue1 (- curv-threshold))
  (setq dot-product (abs (+ (* gradient2 eigenvector1)
                            (* gradient1 eigenvector2))))
  (cond
   ((< dot-product .001)
       (setf (aref terrain-cell-map (- (1- mapsize) ycoord)
                   xcoord)
           (append '(rline ridge)
                   (aref terrain-cell-map
                         (- (1- mapsize) ycoord) xcoord))))
   (t
       (setf (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord)
           (append '(rline no)
                   (aref terrain-cell-map (- (1- mapsize)
                                              ycoord) xcoord))))))
 ((> eigenvalue1 curv-threshold)
  (setq dot-product (abs (+ (* gradient2 eigenvector1)
                            (* gradient1 eigenvector2))))
```

```
(cond
  ((< dot-product .001)
   (setf (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord)
         (append '(rline ravine)
                 (aref terrain-cell-map
                       (- (1- mapsize) ycoord) xcoord))))
  (t
   (setf (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord)
     (append '(rline no)
        (aref terrain-cell-map
              (- (1- mapsize) ycoord) xcoord))))))))
(t
 (setf (aref terrain-cell-map (- (1- mapsize) ycoord) xcoord)
   (append '(rline no)
      (aref terrain-cell-map
         (- (1- mapsize) ycoord) xcoord)))))))))))
```

```
;*******************************************************************************************
;* THIS FILE CONTAINS THE FOLLOWING CONSTANTS
;*
;* 1. curv-threshold - the amount of leadway that the eigenvalue has from zero when
;*    classifying the secondary classification
;* 2. slope-limit - the upper limit of the slope when classifying level terrain
;* 3. unsafe - the upper limit of the slope when classifying safe terrain, any slope
;*    above this value is considered unsafe
;* 4. The various matrices referenced refer to the steps necessary to calculate the
;*    B-matrix-u which allows us to calculate the K-matrix. See thesis narrative for
;*    more information.
;* THIS FILE IS NECESSARY TO EXECUTE (run-program mapsize datafile classfile) LOCATED IN
;* FILE GRID-UTILITIES.LISP
;*******************************************************************************************

(setq curv-threshold 0.01)
(setq slope-limit 2.0)
(setq unsafe 28.0)
(setq A-matrix-u (make-array '(9 6) :initial-contents
                     '((1 -1  1  1 -1  1)
                       (1  0  1  0  0  1)
                       (1  1  1  1  1  1)
                       (1 -1  0  1  0  0)
                       (1  0  0  0  0  0)
                       (1  1  0  1  0  0)
                       (1 -1 -1  1  1  1)
                       (1  0 -1  0  0  1)
                       (1  1 -1  1 -1  1))))
```

```
(setq AT-matrix-u (math:transpose-matrix A-matrix-u))

(setq ATA-matrix-u (math:multiply-matrices AT-matrix-u A-matrix-u))

(setq ATAI-matrix-u (math:invert-matrix ATA-matrix-u))

(setq B-matrix-u (math:multiply-matrices ATAI-matrix-u AT-matrix-u))
```

```
;*************************************************************************************************
;* THIS FILE CONTAINS SEVERAL MISCELLANEOUS FUNCTIONS CALLED BY (run-program mapsi
;* datafile classfile) LOCATED IN GRID-UTILITIES.LISP.  THIS FILE MUST BE INCLUDED IN THE
;* LISP WORLD TO EXECUTE (run-program mapsize datafile classfile).
;*************************************************************************************************

(defun feet-to-meter (feet)
 (* feet 0.3048))


(defun meter-to-feet (meter)
 (* meter 3.281))


(defun rad-to-degree (radians)
 (/ radians 0.0174533))


(defun degree-to-rad (degrees)
 (* degrees 0.0174533))


(defun arctan (y x)
 (cond
  ((or (and (minusp y) (minusp x))
         (and (plusp y) (plusp x)))
   (rad-to-degree (atan (abs y) (abs x))))
  (t (- (rad-to-degree (atan (abs y) (abs x)))))))
```

```
;*******************************************************************************
; THIS FILE CONTAINS GRAPHICS ROUTINES USED BY GRID-UTILITIES.LISP
; IT MUST BE LOADED PRIOR TO EXECUTING (RUN-PROGRAM) IN GRID-UTILITIES.LISP
; INFORMATION ON THE PARAMETERS AND KEYWORDS OF THESE FUNCTIONS CAN BE
; FOUND IN THE MANUAL FOR THE COLOR SYMBOLICS
;*******************************************************************************

(defun make-color-window
       (&rest options &key (superior (color:find-color-screen : create-p t))
       &allow-other-keys)
 (apply #'tv:make-window 'tv:window
        :blinker-p nil
        :borders 2
        :save-bits nil
        :expose-p t
        :label "Demonstration Window"
        :superior superior
        options))

(defvar *color-window* (make-color-window))

(defun make-blue-window (&rest options)
 (apply #'make-color-window
        :erase-aluf(send color:color-screen
                         :compute-color-alu
                         tv:alu-seta 0 0 0.4)
        options))

(defvar *blue-window* (make-blue-window))
```

```
(defvar *orange-alu* (send(send *blue-window* :screen)
                           :compute-color-alu
                           color:alu-x 1.0 0.5 0.0))

(defvar *green-alu* (send(send *blue-window* :screen)
                          :compute-color-alu
                          color:alu-x 0 0.8 0))

(defvar *red-alu* (send(send *blue-window* :screen)
                        :compute-color-alu
                        color:alu-x 0.9 0 0))

(defvar *blue-alu* (send(send *blue-window* :screen)
                         :compute-color-alu
                         color:alu-x 0 0 0.8))

(defvar *white-alu* (send(send *blue-window* :screen)
                          :compute-color-alu
                          color:alu-x 1.0 1.0 1.0))

(defvar *black-alu* (send(send *blue-window* :screen)
                          :compute-color-alu
                          color:alu-x 0 0 0))

(defvar *yellow-alu* (send(send *blue-window* :screen)
                           :compute-color-alu
                           color:alu-x 1.0 .93 0.2))

(defun box (xcoord ycoord shade)
 (send *blue-window* :draw-rectangle 10 10  xcoord ycoord shade))
```

```
(defun read-it (mapsize classfile)
 (setq input-stream (open classfile :direction :input))
 (do ((ycoord 100 (+ ycoord 10)))
   ((= ycoord (+ (* 10 mapsize) 100)))
  (do ((xcoord 25 (1+ xcoord)))
       ((= xcoord (+ mapsize 25)))
   (let ((x (read input-stream)))
       (cond
         ((= x 1)(box (* 10 xcoord) ycoord *black-alu*))
         ((= x 2)(box (* 10 xcoord) ycoord *blue-alu*))
         ((= x 3)(box (* 10 xcoord) ycoord *green-alu*))
         ((= x 4)(box (* 10 xcoord) ycoord *yellow-alu*))
         ((= x 5)(box (* 10 xcoord) ycoord *white-alu*))
         ((= x 6)(box (* 10 xcoord) ycoord *red-alu*))
         ((= x 7)(box (* 10 xcoord) ycoord *orange-alu*))))))
 (close input-stream))
```

LIST OF REFERENCES

[1] Poulos, D. D., "Range Image Processing for Local Navigation of an Autonomous Land Vehicle," M. S. Thesis, Naval Postgraduate School, Monterey, California, September 1986.

[2] Douglas, D. H., "Experiments to Locate Ridges and Channels to Create a New Type of Digital Elevation Model," *Cartographica Vol. 23, No. 4,* (Winter 1986).

[3] Hearn, D. and Baker, M. P., *Computer Graphics* (Prentice-Hall, Englewood, New Jersey, 1986).

[4] Smith, D. B. and Streyle, D. G., "An Inexpensive Real-Time Interactive Three-Dimensional Flight Simulation System," M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1987.

[5] Pavlidis, T., *Algorithms for Graphics and Image Processing* (Computer Science Press, Rockville, Maryland, 1982).

[6] Haralick, R. M., Watson, L. T., and Laffey, T. J., "The Topographic Primal Sketch ," *The International Journal of Robotics Research Vol. 2, No. 1,* (Spring 1983).

[7] Winston, P. H., *Artificial Intelligence* (Addison-Wesley, Reading, Massachusetts, 1984).

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
5.	US Army Combat Developments Experimentation Center (USACDEC) Attention: W. D. West Fort Ord, California 93941	1
6.	Curriculum Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
7.	Professor Robert B. McGhee, Code 52MZ Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	13
8.	Professor Michael J. Zyda, Code 52ZK Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1

9. Mr. M. A. Hunter 1
Route 1, McClure Road
Winchester, Kentucky 40391

10. Ms. Carol Hunter 2
504 Mallard Park
Versailles, Kentucky 40383

11. United States Military Academy 1
Department of Geography & Computer Science
ATTN: Major R. F. Richbourg
West Point, New York 10996-1695

12. Artificial Intelligence Center 1
HQDA OCSA
ATTN: DACS-DMA
The Pentagon RM 1D659
Washington, D.C 20310-0200

13. Director of Research Administration, Code 012 1
Naval Postgraduate School
Monterey, California 93943